Local Storage Guidelines

Introduction

In the Play+ ecosystem, how we handle client-side data is as important as how we display it. This helper is based on the concept of Intentional and Resilient Storage , providing a safe and intelligent abstraction layer over the browser's raw localStorage API.

Direct use of localStorage is discouraged as it's primitive, error-prone, and insecure for sensitive information. playcache is designed specifically for caching non-sensitive data like user preferences, UI state, and temporary application data. This aligns with our core design pillars by creating an Intuitive API that prevents common errors, an Adaptive system that can handle data expiration, and a Distinct , consistent pattern for storage across all projects.

Package Info

The Play+ storage helper is included by default in the Golden Path starter kit. For existing projects, it can be installed via its dedicated package.

Description

Package / Path	Description
Golden Path (Recommended)	Pre-installed (/system/play.cache.ts)
Uplift Path	npm install @playplus/storage
To get started with the Uplift Path, you can also run:	
npx playcache	

Folder Reference

File / Directory	Purpose & Guidelines
system/play.cache.ts	The core storage service. It wraps the localStorage API, providing all helper methods.
config/play.cache.config.json	An optional file for overriding default caching behaviors, such as the key prefix.
src/app/services/playcache.service.ts	Angular service wrapper for reactive caching patterns.

Helper - Pillars Alignment

Pillar	How This Helper Aligns
Intuitive	Primary Pillar: Replaces the error-prone, manual process of using localStorage with a simple, safe, and memorable API.
Adaptive	Supports reactive data patterns through its event system, allowing the UI to adapt to storage changes across tabs.
Distinct	Enforces a single, consistent, and safe way to interact with browser storage across all Play+ applications.

Helper Overview

The playcache helper is a smart and resilient wrapper around the browser's localStorage . Its purpose is to abstract the plumbing of client-side caching, so developers can store and retrieve non-sensitive data without worrying about common pitfalls.

It automates and simplifies:

- Serialization : Automatically runs JSON.stringify() on set and JSON.parse() on get, preventing runtime errors and eliminating boilerplate.
- Key Scoping : Automatically prefixes all keys with a unique identifier to prevent collisions with other browser tabs, third-party scripts, or other apps on the same

domain.
• Data Expiration (TTL) : Supports a "Time-to-Live" for cached items, treating expired data as null.
• Reactivity : Provides an event-based system (on / off) so your app can react to storage changes, even across tabs.
• Error Handling : Graceful handling of storage errors and quota exceeded
• Size Management : Built-in cache size monitoring and cleanup

Config Options

Optional global configuration can be provided in config/play.cache.config.json .

Config Variable	Default Value	Description	Recommended Value
keyPrefix	playcache	The prefix added to all keys to prevent collisions.	Keep default
defaultTtl	null	Default Time-to-Live in seconds. null means no expiration.	null
maxSize	5242880	Maximum cache size in bytes (5MB).	5242880
cleanupInterval	300000	Cleanup interval in milliseconds (5 minutes).	300000

Helper Methods

Core Methods

Method Name	Description	Signature
set	Stores a serializable value in the cache, with optional TTL (in seconds).	set(key: string, value: any, options?: { ttl: number }): void

Method Name	Description	Signature
get	Retrieves a value from the cache. Returns null if key doesn't exist or has expired.	get<T>(key: string): T | null
remove	Deletes a specific item from the cache by its key.	remove(key: string): void
clear	Removes all items from the cache that have the Play+ key prefix.	clear(): void
on	Subscribes to changes for a specific key. Callback is triggered on updates (even across tabs).	on(key: string, callback: (newValue: any) => void): void
off	Unsubscribes a callback for a specific key to prevent memory leaks.	off(key: string, callback: (newValue: any) => void): void

Utility Methods

Method Name	Description	Signature
has	Checks if a key exists and is not expired.	has(key: string): boolean
keys	Gets all keys that match a pattern (supports wildcards).	keys(pattern?: string): string[]
getSize	Gets the size of cached data in bytes.	getSize(): number
cleanup	Cleans up expired items from the cache.	cleanup(): void

Usage Examples

Basic Usage

React: A Custom Hook for Reactive Caching

Angular: A Reactive Service

Angular Service Usage

Component Integration

Additional Info

Why We Created This Helper

Using window.localStorage directly introduces several issues:

- Manual JSON serialization/deserialization leads to bugs.
- No native TTL or expiration.
- Global key collisions.
- No reactivity within the same tab.
- Encourages storing sensitive data insecurely.

playcache solves these by adding scoping, reactivity, TTLs, and a safe abstraction.

Security: For Non-Sensitive Data Only

playcache does not encrypt data and is not safe for sensitive information. Use playguard for secure token and auth storage via HttpOnly cookies.

Data Classification

Data Type	Storage Method	Example
Sensitive	playguard (HttpOnly cookies)	Auth tokens, user credentials
User Preferences	playcache	Theme, language, UI settings
Temporary Data	playcache with TTL	Search results, form data
Static Reference	playcache	Dropdown options, configuration

Caching Strategies

User Preferences

UI State

Temporary Data

Cross-Tab Communication

Best Practices

Do's

- Cache static reference data (e.g., dropdowns, themes, user preferences)
- Use TTLs for temporary data to avoid stale state
- Clear cache intentionally (e.g., on logout)
- Use reactive patterns for UI state that needs to update
- Monitor cache size to prevent quota exceeded errors
- Clean up expired items periodically

Don'ts

- Never cache sensitive data (e.g., PII, tokens, passwords)
- Don't exceed 5MB browser storage limits
- Don't use localStorage directly - always use playcache
- Don't cache dynamic data that changes frequently
- Don't forget to unsubscribe from cache events

Performance Considerations

Cache Size Management

Memory Leak Prevention

Testing

Unit Testing

Developer Checklist

- Am I only storing non-sensitive data (like UI preferences or temporary reference data)?
- For data that can become stale, have I set an appropriate TTL?
- Am I using playcache instead of localStorage directly?
- Am I using on() and off() or reactive patterns if my UI needs to respond to changes?
- Am I clearing or removing cache when appropriate (e.g., logout)?
- Am I monitoring cache size to prevent quota exceeded errors?
- Am I cleaning up expired items periodically?
- Am I unsubscribing from cache events to prevent memory leaks?

Troubleshooting

Common Issues

Cache not persisting across sessions

- Check if the browser supports localStorage
- Verify the key prefix is correct
- Ensure the data is serializable

Cross-tab updates not working

- Verify you're using the on() method correctly
- Check if the storage event is being handled
- Ensure the key prefix matches

Cache size exceeded

- Implement cache cleanup
- Use TTL for temporary data
- Monitor cache size regularly

Performance issues

- Avoid storing large objects
- Use TTL for temporary data
- Clean up expired items